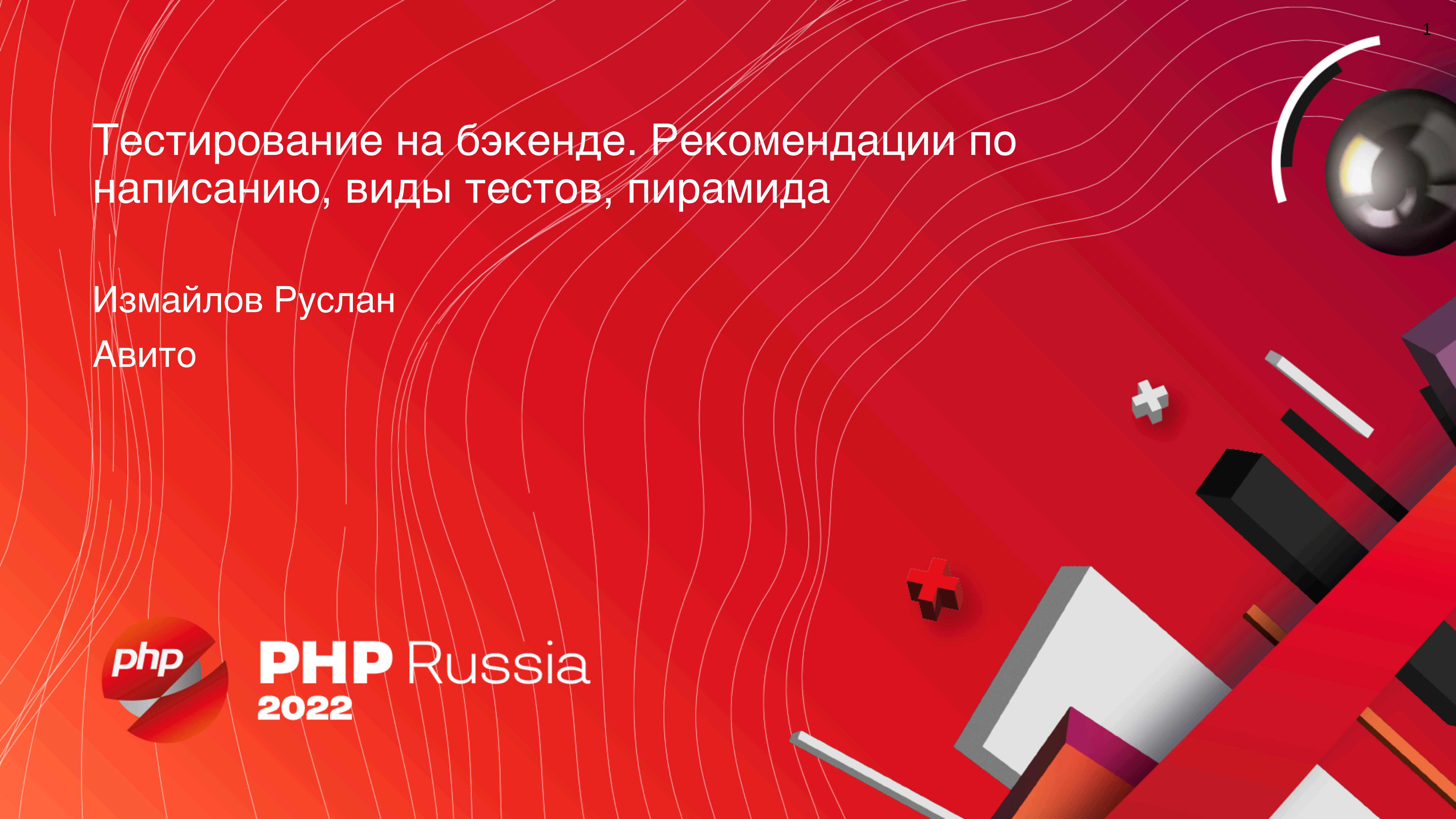
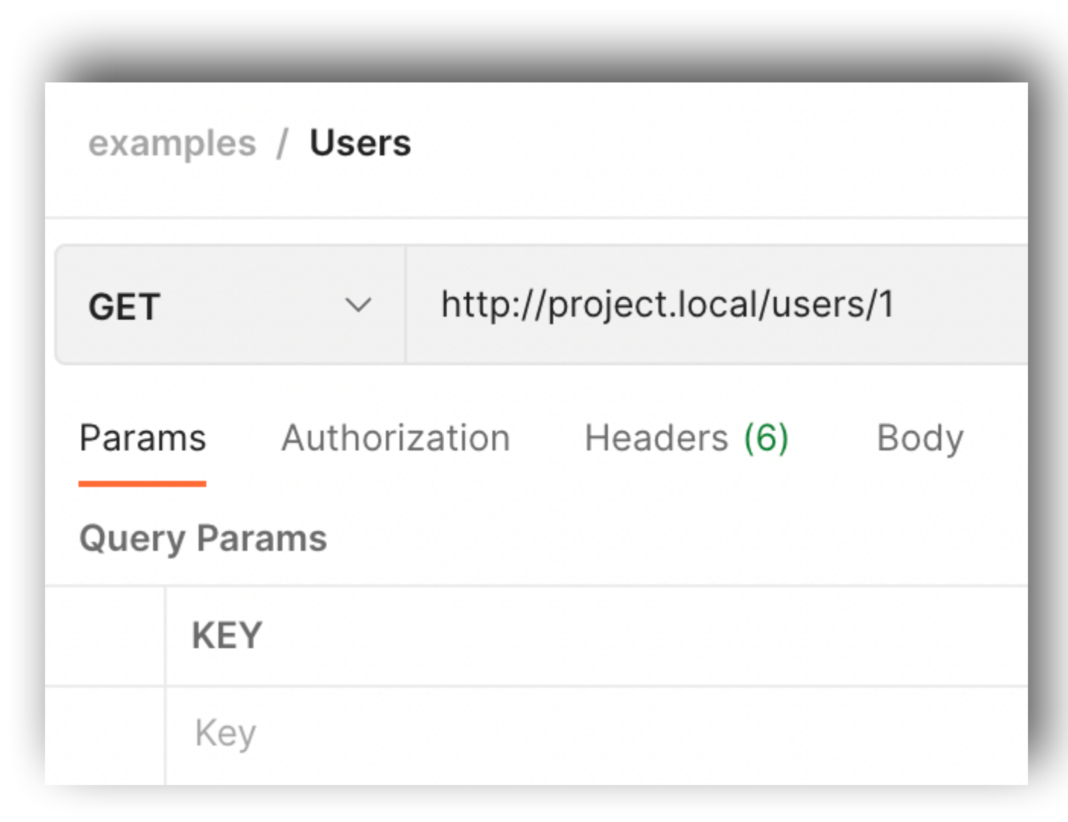

Тестирование на бэкенде. Рекомендации по написанию, виды тестов, пирамида
Измайлов Руслан
Авито
php
PHP Russia
2022

Зачем писать код, который проверяет другой код?



Ручная проверка

```
$I->sendGet('/users/1');
$I->seeResponseCodeIs(HttpCode::OK); // 200
$I->seeResponseIsJson();
$I->seeResponseMatchesJsonType([
    'id' => 'integer',
    'name' => 'string',
    'email' => 'string:email',
    'homepage' => 'string:url|null',
    'created_at' => 'string:date',
    'is_active' => 'boolean'
]);
```

Автоматизация



Все слишком очевидно, тест не нужен

Пфф, что тут тестировать?

```php
function calculate(int $a, int $b): int {
    return $a + $b;
}
```

Простая проверка покажет, что наша функция неидеальная

```php
function testCalculate(): void
{
    $result = calculate( a: PHP_INT_MAX,  b: 1);
    // TypeError: Return value must be of type int,
    // float returned
}
```



Баги и повторяющиеся баги

Работает

```
private ?int $amount;

public function getAmount(): int
{
    if (empty($this->amount)) {
        return self::DEFAULT_AMOUNT;
    }
    return $this->amount;
}
```

Костыльная проверка

```
if (empty($this->amount))
```

Трушная проверка

```
($this->amount === null)
```

И тут приходит 0

```
$this->amount = 0;
```



Каким должен быть идеальный тест?

«Гарантирующий»
что-то проверяет

«Покрывающий»
проверяет все

«Без окружения»
написал и запустил

«Быстрый»
не хочу ждать

«Стабильный»
не зависит от порядка

«Независимый»
рефакторинг — не проблема



Подопытный

```php
class ItemController
{
    private ItemSearchingService $service;

    public function searchAction(Request $request): array
    {
        $items = $this->service->search(
            $request->getSearchQuery(),
            $request->getUserId()
        );
        return $this->formatter->format($items);
    }
}
```

Что хотим протестировать:

- ❑ успешный сценарий: выполнили поиск — получили результат
- ❑ разные поисковые запросы: кириллица/латиница, спец. символы, регистр
- ❑ ценообразование: скидка есть, скидки нет, стоимость доставки



Функциональный тест

Проверяем функциональность целиком — все компоненты в связке

```
public function setUp(): void
{
    $this->initFixtures();
    $this->loginUser();
}
```

```
public function testSearchSucces(): void
{
    $response = $this->tester->sendRequest(
        'item/search',
        ['searchQuery' => 'чистый код']
    );

    $this->assertEquals(['items' => [
        [
            'type' => 'book',
            'title' => 'Чистый код',
            'price' => 2650,
        ]
    ]], $response);
}
```



Преимущества и недостатки

«Гарантирующий»
в тесте задействованы боевые компоненты

«Покрывающий»
все не проверим

«Без окружения»
нужна тестовая среда

«Быстрый»
инициализация-запрос-ответ

«Стабильный»
отказ одного из компонентов

«Независимый»
поломается только при изменении контракта



Рекомендации

- На каждую точку входа (endpoint) хотя бы один тест (успешный сценарий)
- Тест-кейсы формировать в первую очередь на основе контракта



Подопытный

```php
class ItemController
{
    private ItemSearchingService $service;

    public function searchAction(Request $request): array
    {
        $items = $this->service->search(
            $request->getSearchQuery(),
            $request->getUserId()
        );
        return $this->formatter->format($items);
    }
}
```

```php
class ItemSearchingService
{
    private ItemRepositoryInterface $itemRepository;
    private UserRepositoryInterface $userRepository;

    public function search(string $searchQuery, int $userId)
    {
        $user = $this->userRepository->get($userId);
        $items = $this->itemRepository->find($searchQuery);
        foreach ($items as $item) {...}

        return $items;
    }
}
```



Подопытный

```php
class ItemSearchingService
{
    private ItemRepositoryInterface $itemRepository;
    private UserRepositoryInterface $userRepository;

    public function search(string $searchQuery, int $userId)
    {
        $user = $this->userRepository->get($userId);
        $items = $this->itemRepository->find($searchQuery);
        foreach ($items as $item) {...}

        return $items;
    }
}
```

Что хотим протестировать:

- [x] успешный сценарий: выполнили поиск — получили результат
- [ ] разные поисковые запросы: кириллица/латиница, спец. символы, регистр
- [ ] ценообразование: скидка есть, скидки нет, стоимость доставки



Интеграционный тест

Проверяем интеграцию бизнес-логики с инфраструктурными компонентами

```
public function setUp(): void
{
    $this->initDatabase();
}
```

```
public function testFindShouldBeCaseInsitive(): void
{
    $actualItems = $this->itemRepository->find('чИстЫй КОД');

    $this->assertEquals($this->getExpectedItems(), $actualItems);
}

public function testFindWithLatinLetters(): void{...}

public function testFindWithSpecialCharacters(): void{...}
```



Преимущества и недостатки

«Гарантирующий»
взаимодействие с инфраструктурой работает

«Покрывающий»
большинство кейсов

«Без окружения»
нужна тестовая среда

«Быстрый»
запрос в БД

«Стабильный»
другой тест испортил фикстуры

«Независимый»
поломается только при изменении контракта



Рекомендации

- Покрываем интеграционным тестами все «стыки» с инфраструктурой:
  - репозиторий
  - http-клиент
  - и другие i/o-компоненты
- Сами компоненты делаем максимально «тонкими»



Как тестировать http-клиенты

Используем мок-серверы




```
"pairs": [
  {
    "request": {...},
    "response": {"status": 200...}
  },
  {
    "request": {...},
    "response": {"status": 500...}
  }
],
```



Подопытный

```php
class ItemSearchingService
{
    private ItemRepositoryInterface $itemRepository;
    private UserRepositoryInterface $userRepository;

    public function search(string $searchQuery, int $userId)
    {
        $user = $this->userRepository->get($userId);
        $items = $this->itemRepository->find($searchQuery);
        foreach ($items as $item) {...}

        return $items;
    }
}
```

```php
foreach ($items as $item) {
    $delivery = $this->getDelivery($item, $user);
    $dicount = $this->getDiscount($item, $user);
    $totalPrice = $item->getBasePrice()
        + $delivery
        - $dicount;
    $item->setTotalPrice($totalPrice);
}
```



Подопытный

```
private ItemRepositoryInterface $itemRepository;
private UserRepositoryInterface $userRepository;

public function search(string $searchQuery, int $userId)
{
    $user = $this->userRepository->get($userId);
    $items = $this->itemRepository->find($searchQuery);
    foreach ($items as $item) {
        $delivery = $this->getDelivery($item, $user);
        $dicount = $this->getDiscount($item, $user);
        $totalPrice = $item->getBasePrice()
            + $delivery
            - $dicount;
        $item->setTotalPrice($totalPrice);
    }

    return $items;
}
```

Что хотим протестировать:

- [x] успешный сценарий: выполнили поиск — получили результат
- [x] разные поисковые запросы: кириллица/латиница, спец. символы, регистр
- [ ] ценообразование: скидка есть, скидки нет, стоимость доставки



Unit-тест

Проверяем отдельный компонент без взаимодействия с инфраструктурой

```php
private function createTestingInstance(): ItemSearchingService
{
    $userRepositoryMock = $this->createConfiguredMock(
        UserRepositoryInterface::class,
        ['get' => new TestUserWithPersonaDiscount()]
    );
    $itemRepositoryMock = $this->createConfiguredMock(
        ItemRepositoryInterface::class,
        ['find' => new TestItemCollection()]
    );

    return new ItemSearchingService(
        $userRepositoryMock,
        $itemRepositoryMock
    );
}
```

```php
public function testSearchWithDiscount(): void
{
    $itemSearchingService = $this->createTestingInstance();

    $actualItems = $itemSearchingService->search();

    $this->assertEquals($expectedItems, $actualItems);
}

public function testSearchWithoutDiscount(): void{...}
public function testSearchWithDelivery(): void{...}
```



Преимущества и недостатки

«Гарантирующий»
компонент работает ожидаемо

«Покрывающий»
можно покрыть все кейсы

«Без окружения»
ничего не нужно

«Быстрый»
i/o-операций нет

«Стабильный»
изолирован от внешних факторов

«Независимый»
рефакторинг уничтожит тест



Улучшим независимость

Пример простого рефакторинга

```php
private ItemRepositoryInterface $itemRepository;
private UserRepositoryInterface $userRepository;

public function search(string $searchQuery, int $userId): ItemCollection
{
    $user = $this->userRepository->get($userId);
    $items = $this->itemRepository->find($searchQuery);
    foreach ($items as $item) {
        $deliveryPrice = $this->calculateDeliveryPrice($item, $user);
        $dicount = $this->getDiscount($item, $user);
        $totalPrice = $item->getBasePrice() + $deliveryPrice - $dicount;
        $item->setTotalPrice($totalPrice);
    }

    return $items;
}
```

```php
private ItemRepositoryInterface $itemRepository;
private UserRepositoryInterface $userRepository;
private Pricer $pricer;

public function search(string $searchQuery, int $userId): ItemCollection
{
    $user = $this->userRepository->get($userId);
    $items = $this->itemRepository->find($searchQuery);
    foreach ($items as $item) {
        $totalPrice = $this->pricer->calulateTotalPrice($item, $user);
        $item->setTotalPrice($totalPrice);
    }

    return $items;
}
```



Улучшим независимость

Наш тест требует доработки

```
private function createTestingInstance(): ItemSearchingService
{
    $userRepositoryMock = $this->createMock(UserRepositoryInterface::class);
    $itemRepositoryMock = $this->createMock(ItemRepositoryInterface::class);
  + $pricerMock = $this->createMock(Pricer::class);

    return new ItemSearchingService(
        $userRepositoryMock,
        $itemRepositoryMock,
      + $pricerMock
    );
}
```



Улучшим независимость

Подготовим стабы

```
class UserRepositoryStub implements UserRepositoryInterface
{
    public const
        USER_WITH_DISCOUNT = 1,
        USER_WITHOUT_DISCOUNT =2;

    public function get(int $userId): User
    {
        switch ($userId) {
            case self::USER_WITH_DISCOUNT:
                return new UserWithDiscount();
            case self::USER_WITHOUT_DISCOUNT:
                return new UserWithoutDiscount();
        }

        throw new UserNotFountException();
    }
}
```

```
class ItemRepositoryStub implements ItemRepositoryInterface
{
    public const
        QUERY_CLEAN_CODE = 'чисты код',
        QUERY_BOOKS = 'книги',
        QUERY_TABLE = 'стол';

    public function find(string $query): ItemCollection
    {
        switch ($query) {...}

        return new ItemCollection([]);
    }
}
```



Улучшим независимость

Настроим контейнер зависимостей

```
if ($environment === 'test') {
    return [
        UserRepositoryInterface::class => UserRepositoryStub::class,
        ItemRepositoryInterface::class => ItemRepositoryStub::class,
    ];
}

return [
    UserRepositoryInterface::class => UserRepository::class,
    ItemRepositoryInterface::class => ItemRepository::class,
];
```



Улучшим независимость

Посмотрим на результат

Было

```
private function createTestingInstance()
{
    $userRepositoryMock = $this->createMock(
        UserRepositoryInterface::class
    );
    $itemRepositoryMock = $this->createMock(
        ItemRepositoryInterface::class
    );
    return new ItemSearchingService(
        $userRepositoryMock,
        $itemRepositoryMock
    );
}
```

Стало

```
private function createTestingInstance()
{
    return $this->container->get(
        ItemSearchingService::class
    );
}
```



Преимущества и недостатки

«Без окружения»

ничего не нужно

«Независимый»

рефакторинг уничтожит тест

«Без окружения»

нужны стабы

«Независимый»

поломается только при изменении контракта



Рекомендации

- Покрываем всю бизнес-логику
- Дополняем интеграционные тесты
- Больше стабов, меньше моков
- Простые компоненты — простые тесты



Stub vs Mock

Stub

```
$this->createStub(UserRepository::class)
    ->method('get')
    ->willReturn(new User());
```

Тестовая реализация

Используем для читающих методов

Mock

```
$this->createMock(UserRepository::class)
    ->method('get')
    ->expects($this->once())
    ->willReturn(new User());
```

Ожидаем вызова метода

Используем для пишущих методов

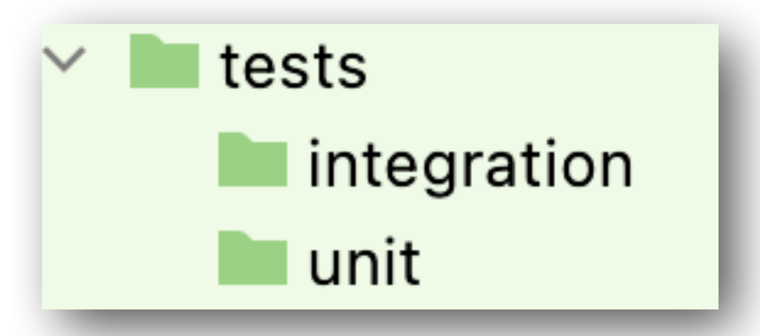


Unit… Integration… Зачем?

Unit:
- нужен только код
- можно легко запускать параллельно

Integration:
- нужна инфраструктура
- параллельно — сложно

Делим на директории



Размечаем аннотациями

```
/**
 * @group integration
 */
class ItemRepositoryTest
```

CI/CD:
- …
- Стат. анализ
- Unit-тесты
- Integration-тест
- …



Сравним тесты

	Функциональный	Интеграционный	Юнит
Гарантирующий	🟢	🟠	🔴
Покрывающий	🔴	🟠	🟢
Без окружения	🔴	🔴	🟢
Быстрый	🔴	🟠	🟢
Стабильный	🔴	🟠	🟢
Независимый	🟢	🟢	🔴



Пирамида тестирования

Функциональные — Все в связке — ок

Интеграционные — Инфраструктура — ок

Юниты — Бизнес-логика — ок



Заключение

- Тесты стоит писать
- Баги покрываем всегда
- Тест-кейсы формируем на основе контракта
- Жонглируем пирамидой
- Хороший код порождает хороший тест и наоборот



Измайлов Руслан
Авито

izmaylov2207



Обратная связь
и комментарии по
докладу по ссылке

